L2/25-179

Text Terminal Working Group report

Author: Fraser Gordon (chair)

The Text Terminal Working Group (TTWG) has been meeting on a monthly basis. Among regular attendees, we have a good cross section of expertise, including authors of terminal emulators, text rendering libraries and text shapers. More participants are very welcome and we operate a wide membership policy: we will invite anyone with interest and expertise in related fields to the working group, either to actively participate or observe the work in progress and offer feedback.

Infrastructure for the private use of WG has been set up on Github (with thanks to the infrastructure team!); we had intended to make the Github repo public but the licensing situation becomes complicated and we are deferring that for now.

At this stage, we have extensively discussed the problem domain and are working towards a high-level technical design.

In order to simplify our task, we intend to use the *Recommended Scripts* table in UAX 31 as a guide to prioritise support for widely-used scripts. We have informally divided these scripts into groups based on their layout needs. As it is unlikely that every script will be supported in the first release, we anticipate that further releases will be needed to expand script support. Note that this would preclude offering a broad stability guarantee: it is expected that the display of many strings will change over time.

Some recommended scripts do not have broad evidence of use of fixed-width text in terminals or even with typewriters (e.g. Bengali). Without this, we do not have examples of how users would like these scripts to work in terminals. As such, the rules for scripts in this situation are going to need iteration, based on user feedback. We would welcome contributions from anyone with knowledge of prior art for fixed-width Indic text.

Because of these uncertainties, versioning is a priority. Applications and terminals need to have the ability to communicate to each other the level of support that is available. The widely-used escape sequence mechanism seems adequate for this purpose. We have explicitly rejected any mechanisms that require font-dependent feedback as there are real-world scenarios where the application (or sometimes even the terminal) does not have access to the font in use. In other words, no layout rules can depend on font data.

As an approximation, almost no existing applications that write to terminals are going to be updated to implement any algorithms that we propose, since they write strings to the terminal and expect the terminal to do the right thing. This would include `cat` and similar utilities but also every application that logs errors to the terminal. This constrains us to

ensure that the default behaviour in the absence of negotiation does something appropriate for reasonable cases.

Applications and layout libraries with complex needs also exist (terminal-based text editors are a prime example). These need to work with the terminal to provide appropriate display and editing behaviour. To accomplish this, the division of responsibilities between the two sides needs to be clear and explicit. Versioning is necessary for this to ensure a consistent set of rules is used.

Our recommendations will need to extend beyond display of text in terminals (the output side). Handling of input also needs to be specified. How terminals should respond to cursor movement, either by the user or by escape sequences, should be defined; an example of complexity here is cursor positioning within a grapheme cluster. There is a large area of overlap here with UAX 9 (the Unicode Bidirectional Algorithm) and we anticipate that support for UAX#9, at least in part, will be a prerequisite for implementations. The details of this need further discussion.

Similar to the bidi algorithm, complications arise with respect to display order versus logical order. The terminal needs to be aware of display order for obvious reasons but logical order may also be necessary to efficiently support operations like copy-paste. This is likely to be another area of difference between default behaviour and complex application behaviour.

This simple/complex split is going to be a key part of the WG's work and may prove to be as difficult to get right as devising the layout rules themselves. The group has representation from terminal implementers but more expertise from complex applications will be needed.